Are You Still There?
— A Lightweight Algorithm To Monitor Node Presence in Self-Configuring Networks —

Henrik Bohnenkamp[a], Johan Gorter[a], Jarno Guidi[b] and Joost-Pieter Katoen[*,a,c]

[a] University of Twente, 7500 AE Enschede, the Netherlands

[b] Philips Research, Prof. Holstlaan 4, 5656 AA Eindhoven, the Netherlands

[c] RWTH Aachen University, 52056 Aachen, Germany

Abstract

This paper is concerned with the analysis and redesign of a distributed algorithm to monitor the availability of nodes in self-configuring networks. The simple scheme to regularly probe a node—"are you still there?"—may easily lead to over- or underloading. The essence of the algorithm is therefore to automatically adapt the probing frequency. We show that a self-adaptive scheme to control the probe load, originally proposed as an extension to the UPnP^{TM} (Universal Plug and Play) standard, leads to an unfair treatment of nodes: some nodes probe fast while others almost starve. An alternative distributed algorithm is proposed that overcomes this problem and that tolerates highly dynamic network topology changes. The algorithm is very simple and can be implemented on large networks of small computing devices such as mobile phones, PDAs, and so on.

Keywords: discrete-event simulation, distributed algorithms, formal specification, performability evaluation, plug-and-play networks, self-configuring networks

1. Introduction

This paper is concerned with the *membership management* of highly dynamic, self-configuring networks. In particular, we are concerned with the analysis of a distributed algorithm [1] that is aimed to maintain (and to some extent disseminate) *up-to-date* information about the presence (or absence) of devices (also called nodes). That is to say, the distributed algorithm allows for the monitoring of the availability of a node by other nodes. Normally, when a node goes off-line, it informs other nodes by sending a bye-message, but if it suddenly becomes unavailable, no such indication is sent, and the studied protocol comes into play. An important requirement is that the absence of nodes should be detected quickly (e.g., in the order of one second) while avoiding to overload nodes. As the basic mechanism of the membership management algorithm is to simply regularly check (i.e., probe) whether a node is still present, we refer to it as the probe protocol. Related protocols are failure detection and monitoring protocols. For a survey we refer to [6].

In our setting there are two types of nodes, *devices* and *control points*, and only the failure of a single type of node is relevant (devices). The probe protocol considered in this paper shares many aspects with the newscast computing approach [11]: it maintains up-to-date (membership) information in a self-organizing way, without any central intervention, in a dynamically changing and large-scale distributed environment. In particular, it continues to operate properly without manual intervention under the—according to varying patterns—joining and (un)intentional leaving of devices.

This paper describes two probe protocols for membership management and discusses analysis results obtained by means of discrete-event simulation. The first protocol, which we called the *self-adapting probe protocol* (SAPP), was proposed in [1] and can be implemented as an (proprietary) extension of the UPnP^{TM} (Universal Plug and Play) standard. The second protocol is our contribution and overcomes some of the problems of [1]. In the following, we will call this protocol the *device-controlled probe protocol* (DCPP).

The simplest scheme one could consider is to regularly probe a device—"are you still there?". This scheme, however, easily leads to over- or underloading of devices. The essence of both algorithms is therefore to automatically adapt the probing frequency. We show that the self-adaptive scheme of SAPP to control the probe load of a probed device leads to an unfair treatment of control points (CPs): some CPs probe fast while others almost starve.

The main contribution of this paper is, first, the analysis

Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)
0-7695-2282-3/05 $20.00 © 2005 IEEE

of the SAPP [1], and second, our alternative DCPP algorithm that overcomes the mentioned unfairness problem. The simplicity of DCPPallows for the implementation on large networks of small computing devices such as mobile phones, PDAs, and so on.

The paper is organized as follows. Section 2 describes the SAPP as proposed by Bodlaender *et al.* [1] and focuses in particular on the self-adaptive scheme to regulate the probe frequency. Section 3 discusses the analysis results obtained by discrete-event simulations. Section 4 and 5 describe the alternative probe protocol DCPP and its analysis results. Finally, Section 6 concludes.

2. The Self-Adaptive Probe Protocol

In this section we describe the self-adaptive probe protocol SAPP, as proposed in [1]. Two types of nodes are distinguished: simple nodes (*devices*) and somewhat more intelligent ones, called *control points* (CPs). The basic protocol mechanism is that a CP continuously probes a device that in turn replies to the CP, if it is still present. The CP adapts the probing frequency automatically in case a device tends to get over- or underloaded. The CPs are dynamically organized in an overlay network by letting the device, on each probe, return the ids of the last two (distinct) processes that probed it. On detecting the absence of a device, the CP uses this overlay network to inform all CPs about the leave of the device rapidly. This information dissemination phase of the protocol is not considered in this paper.

Device behavior. A device maintains a probe-counter *pc* that keeps track of the number of times the device has been probed so far. On receipt of a probe, this counter is incremented by the natural Δ, and a reply is sent to the probing CP with as parameter the (just updated) value of *pc*. The returned value of *pc* is used by CPs to estimate the load of the device. Note that *pc* might have been increased also due to probing of other CPs. Therefore it is not enough to just send the value Δ back to the currently probing CP. Δ is device-dependent, and typically only known to the device. Its value may change during execution. A CP can therefore not distill the actual probing frequency of a device, but only an estimate, which we call the *experienced probe load*, denoted as L_{exp}. L_{exp} is computed from the received values for *pc*.

The factor Δ is used by a device to control its load. For larger Δ, CPs consider the device to be more (or even over-) loaded sooner, and will adjust (i.e., lower) their probing frequency accordingly resulting in a lower probe-load at the device. This works as follows. We assume a so-called *ideal probe load*, denoted by L_{ideal}, known to all CPs and devices. L_{ideal} is a reference constant only and must have an high value. CPs use the relation between L_{exp} and L_{ideal} to adapt their ping frequency.

In addition, a device is assumed to have a private *nominal probe load* L_{nom}, representing the actual load, measured in probes per second, the device can or wants to maintain during normal operation. Defining now $\Delta \geqslant \lceil L_{ideal}/L_{nom} \rceil$, enables the device to slow down the CPs (assuming $L_{ideal} \gg L_{nom}$) as it informs the CPs that it is a factor Δ more "busy" than is really the case. This allows the device to make sure that only a fraction of its computational resources is used for answering probes and can devote the rest of its resources to its primary tasks. If the device finds that it is getting to many probes, it can, say, double its value of Δ. As a consequence, the CPs will consider the device more busy and adapt their respective probing frequencies accordingly. The probe load of the device will, in this example, eventually drop to one half of its previous value. Note that the optimal probe frequency for a CP with k CPs in total is L_{nom}/k, but as neither k nor L_{nom} are known to a CP, the adaptive mechanism described below must take care of keeping the probing frequencies of the CPs reasonably close to this optimum.

CP behavior. The behavior of a CP is more intricate. The basic mechanism for communicating with a device is a bounded retransmission protocol (à la [8]): a CP sends a probe ("are you still there?"), and waits for a reply. In absence of a reply during a certain timeout period, it retransmits the probe (cf. Fig. 1). Otherwise, the CP considers the reply as a notification of the presence of the device, and continues its normal operation. Probes are retransmitted maximally three times. If on none of the four probes a reply is received, the CP considers the device to have left the network. The protocol allows to distinguish between the timeout value TOF after the first probe and the timeout value TOS after the other (maximally three) probes. Typically, TOS $<$ TOF. The reason for different values is that, if the first probe remains unanswered, the probability that the device is really absent is already quite high. It is therefore reasonable to send the three other probes in more rapid succession, in order to shorten the time until absence of the device is detected.

In all simulation studies in this paper TOF equals 0.022 (i.e., two times the round-trip delay of the considered network + the maximal computation time of the device), and TOS equals 0.021 (1 times round-trip delay + maximal computation time of the device).

Adapting the probing frequency. Let us now consider the mechanism for a CP to determine the probing frequency of a device. A *probe cycle* starts with a probe and

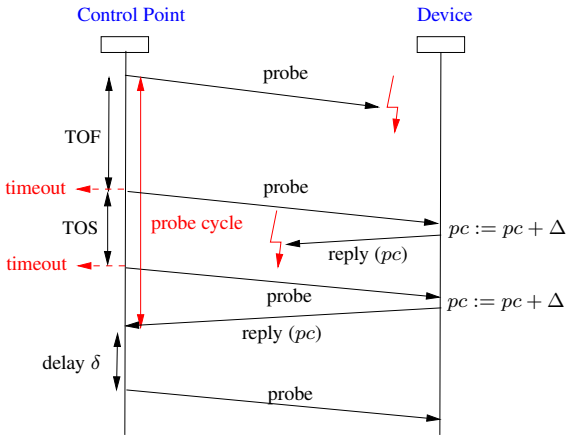

Figure 1. Elementary protocol mechanism

ends with either a reply (a successful probe) or with a timeout after three retransmissions of the probe (an unsuccessful probe). Let δ be the delay between two consecutive probe cycles. There is a minimal and maximal inter-probe-cycle delay, i.e., a CP has to obey $\delta_{min} \leqslant \delta \leqslant \delta_{max}$ for constants δ_{min} and δ_{max} with $\delta_{max} \gg \delta_{min}$. The value of δ is adapted to keep the probe load L_{exp} as perceived by a CP "close to" the ideal probe load: $\frac{1}{\beta} \cdot L_{ideal} \leqslant L_{exp} \leqslant \beta \cdot L_{ideal}$ for $\beta > 1$. Note that β is just a constant. Assume the CP receives a reply on a probe with probe-count *pc* at time t. (In case of a failed probe, the time at which the retransmitted probe has been sent is taken.) The next reply is received at time $t' > t$, and let *pc'* be its returned probe-count. $t'-t$, thus, is the time delay between two successive successful probes. The experienced probe load is now given by $L_{exp} = (pc' - pc)/(t' - t)$. The inter-probe-cycle delay δ is adapted according to the following scheme, where δ' and δ refer to the new and current value of δ, respectively:

$$\delta' = \begin{cases} \min\left(\alpha_{inc} \cdot \delta, \delta_{max}\right) & \text{if } L_{exp} > \beta \cdot L_{ideal} \\ \max\left(\frac{1}{\alpha_{dec}} \cdot \delta, \delta_{min}\right) & \text{if } L_{exp} < \frac{1}{\beta} \cdot L_{ideal} \\ \delta & \text{otherwise} \end{cases} \quad (1)$$

where $\alpha_{inc} > 1$ and $\alpha_{dec} > 1$.

This scheme is justified as follows. In case the just perceived probe load L_{exp} exceeds the maximal load, the delay δ is extended (by a factor $\alpha_{inc} > 1$) with the aim to reduce the load. As δ should not exceed the maximal delay δ_{max}, we obtain the first clause of the above formula. This rule thus readjusts the probing frequency of a CP in case the number of CPs (probing the device) suddenly increases. If L_{exp} is too low, the delay is shortened in a similar way while obeying $\delta_{min} \leqslant \delta$. The second rule thus readjusts the probing frequency of a CP in case the number of CPs (probing the device) suddenly decreases. In all other cases, the load is between the maximal and minimal load, and there is no need to adjust the delay. Note that the maximal frequency at which a CP may probe a device—given that the protocol is in a stabilized situation—is given by $\min(\frac{1}{\delta_{min}}, \beta \cdot L_{nom})$.

3. Modeling and Analysis

The SAPP has been modeled using MODEST, a modeling formalism for stochastic and timed systems [3, 7]. MODEST is a formalism that is aimed to support (i) the modular description of reactive system's behavior while covering both (ii) functional and (iii) non-functional system aspects such as timing and quality-of-service constraints in a single specification. Elaborate descriptions of the MODEST language can be found in [3, 7]. MODESTs compositionality makes it usable in a wide range of different application areas, e.g., in [4] it is used to assess the quality of schedules for a lacquer-production plant in the presence of stochastic failures.

The entire model of the SAPP consists of the parallel composition of a number of CPs, one device, and a network process. We consider only one device since devices and the respective connected CPs in range can be considered as independent from other devices/CPs. Due to lack of space, the models will not be described here. They are available from [10].

Steady-state analysis. The MODEST models were simulated with the MÖBIUS tool [2, 9]. In order to obtain insight into the protocol's behavior we first carried out a steady-state simulation using the batch-mean technique and confidence interval 0.1 with a confidence level of 0.95. The values for the parameters that determine the extent to which to enlarge or shorten the delay δ between probe cycles are given by [1]: $\alpha_{inc} = 2$ and $\alpha_{dec} = \frac{3}{2}$. Other important parameter values that were used for the steady-state simulation were: $\beta = \frac{3}{2}$, $L_{ideal} = 10^6$ and $L_{nom} = 10$ (yielding $\Delta = 10^5$), $\delta_{min} = 0.02$ and $\delta_{max} = 10$. Furthermore, it has been assumed that there is a single device, and $k = 20$, i.e., 20 CPs are continuously present. In a stabilized situation, one would expect the CPs probing at a frequency of $L_{nom}/k = 0.5$. To avoid buffer overruns, the network buffer size has been fixed to 20,000 elements. The network delay has been modeled as a uniform probabilistic choice between three modes of operation: a slow, a medium and a fast mode. We have experimented with several other types of networks, and obtained similar phenomena for all of them.

The analysis focused on the probe frequency $\frac{1}{\delta}$ of the CPs. The simulation revealed that indeed network buffer overflow is a seldom phenomenon as the average buffer length is very small (≈ 0.004). Surprisingly, however, the mean delay of almost all CPs was about 10.0, whereas two CPs had a delay of only 0.4. Both values are far away from the optimal delay, $k/L_{nom} = 2$. Whereas the self-adaptive mechanism should—surely in a static system situation—lead to an equal spreading of the probe frequencies, this is thus apparently not the case. Secondly, some CPs have a high variance in their computed delays, whereas others have only minimal variation. The most extreme case is a CP with a mean delay of 8 and a variance of about 13.5. Despite this abnormal behavior of the CPs, the device load is quite good (i.e., it is near to $L_{nom} = 10$, and has a low variance). These unexpected steady-state simulation results motivated a set of more detailed simulations in order to obtain a better insight into these phenomena.

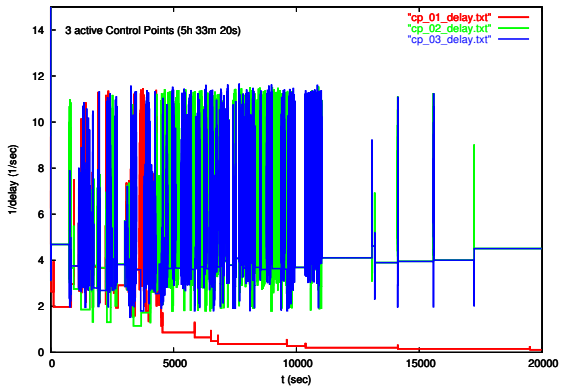

Figure 2. Probe frequencies for 3 CPs

Transient simulation. Various transient simulations have been carried out that simulated the initial 20,000 seconds of the SAPP. All studied configurations consist of a single device and a number of CPs. Whereas for one or two CPs the probe frequencies were balanced and exhibit almost no variance as expected, for three CPs this is not the case: Fig. 2 shows the probe frequencies of the 3 CPs (y-axis) versus the elapsed time (x-axis). Note that after a short initial phase, one CP is probing less and less frequent, and is not recovering from this (undesired) situation. Another observation is that although the remaining two CPs tend to "stabilize" their probing frequencies, there remains to be a rather high variance in their frequencies. Simulations of other scenarios with different numbers of CP show similar behavior: some CPs probe rather fast, whereas others are probing slower and slower, and do not recover from this situation. Similarly, high variances in the individual probe frequencies of a single CP occur. Fig. 3 shows for 7 arbitrary CPs out of a collection of 20 CPs how their probing frequencies evolve over a short interval of 1 minute.

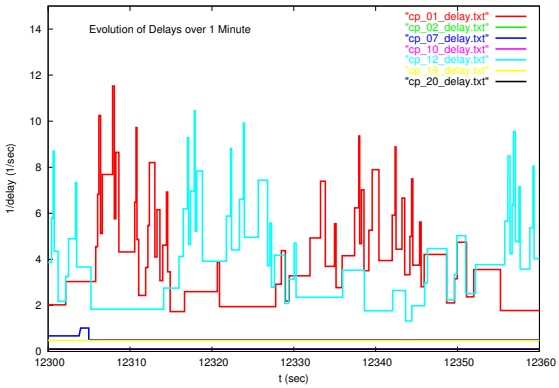

Figure 3. Probe frequencies for 7 (out of 20) CPs for 1 minute

Fig. 4 shows the effect for two CPs that start in a configuration of 20 present CPs of which 18 suddenly leave at the same time. Whereas in a static scenario with just two CPs, their frequencies are equal, we see that in this dynamic scenario, there is neither a load balance between the CPs nor a low variance.

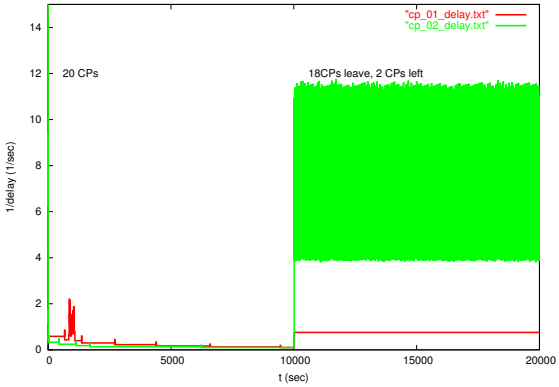

Figure 4. Suddenly 18 out of 20 CPs leave

The above analysis revealed (at least) two undesired phenomena: the variance in probing frequencies of (individual and between) CPs may be extremely large, and CPs can even be completely starving without the possibility to recover from this situation. An important source of these undesired phenomena is that the experienced probe load

L_{exp} of a CP may be misleading. In particular, a CP cannot distinguish between many CPs that probe the same device at medium rate, or a few CPs probing the device at high frequency. In both cases, the device tends to be overloaded, and the difference $pc' - pc$ is large, resulting in a lowering of the CP's probe frequency. Moreover, if the device load drops below the threshold L_{nom}, the CPs with a higher pinging frequency will detect this faster. Due to the greedy nature of the protocol they will therefore increase their respective probe frequencies, thus increasing the probe load of the device. CPs with a low frequency are therefore with high probability too late to get their share of the available bandwidth. The SAPP as described here has therefore an inherent fairness problem. These considerations form the basis for the next protocol.

4. The Device-Controlled Probe Protocol

The protocol described in this section is intended to overcome the fairness problem encountered just above. Like before, there are two types of nodes (devices and CPs) and the basic protocol mechanism is that CPs continuously probe a device to check its presence. Instead of keeping track of the number of times the device has been probed so far, a device simply schedules when a probing CP is allowed to probe it again. Therefore, we call this protocol *device-controlled* (DCPP). The device replies to a probe with a delay value indicating to the respective CP how long it has to wait before it may probe again. To facilitate this scheme, the device keeps track of the time instant at which it wants to be probed again.

Device behavior. A device remembers the time instant nt for which the last pinging CP has been scheduled to probe again. Initially, $nt{=}0$. On receipt of a new probe from a CP at time t, this CP is scheduled to probe again at time $nt'{>}nt$. nt' is computed as $nt'{=}\max\{nt,t\}{+}\Delta(nt,t)$. nt' is the time instant where the pinging CP is scheduled for its next probe. A reply is sent to the probing CP with as parameter the *delay* $nt'{-}t$. $nt'{-}t$ is the time the CP has to wait until is allowed to probe again. $\Delta(nt,t)$ is a function determining the dealy to the next probe. It is used to schedule the probing CPs such that the device load is kept within proper limits. In the following we define $\Delta(nt,t)$.

As in the adaptive protocol, we assume that the device has a certain maximal probe load, L_{nom} which it is able or willing to cope with. We define $\delta_{min} = 1/L_{nom}$. Additionally, we assume that all CPs do only want to ping with a maximal frequency f_{max}. We define $d_{min} = 1/f_{max}$. The following two constraints should be fulfilled in order to keep the load of the device within proper limits. Assume again that a probe has just arrived at time t. Then: (i) $\Delta(nt,t) \geqslant \delta_{min}$, which asserts that two consecutive probes are at least δ_{min} time units apart, and (ii) $nt{-}t{+}\Delta(nt,t) \geqslant d_{min}$, which states that the current CP does not need to probe earlier than $t{+}d_{min}$ again. Summarizing, we have $\Delta(nt,t) \geqslant \max\{\delta_{min}, d_{min}{-}(nt{-}t)\}$. In order to be as fast as possible without violating the both constraints, we choose in the sequel $\Delta(nt,t) = \max\{\delta_{min}, d_{min}{-}(nt{-}t)\}$. Note that the two requirements (i) and (ii) are exactly the constraints needed to keep the probe load below L_{nom}, and to ensure that no CP has to probe more often than it wants to.

CP behavior. The CP behavior is, compared to the SAPP, much simpler. The CP shows the same behavior with respect to the probing and re-probing of a device, however, the delay between two probe cycles is now directly determined by the device. Each reply to a probe is accompanied with a delay d, as computed by the device. On receipt of such reply, the CP sets a timer and waits until d time-units have passed before it initiates the next probe cycle.

5. Analysis of DCPP

We have analyzed the DCPP, again with the MODEST-MÖBIUS tool tandem. Due to its deterministic nature, the protocol ensures that once a situation is reached where the number of probing CPs does not change, the device has a probe load of L_{nom}, and the probe frequency is nearly the same for all CPs. Therefore, the protocol has a big advantage over the adaptive protocol described in Section 2, and is even computationally simpler.

In a dynamic scenario, where CPs join or leave the network in a nondeterministic fashion, and perhaps even in bursts, it is less obvious to determine whether the protocol can meet its expectations. CPs that join the network and start probing the device are unaware of the current schedule laid out by the device. Their entrance will, therefore, disturb the neat pattern of the probe schedule, yielding a (temporarily) increase of the device's load. Simulations have been carried out to find out how the device load is affected in a dynamic environment. These studies concentrated on the average load and its variance. Consider a worst case scenario, where the number of active CPs is uniformly chosen from the set $\{1, \ldots, 60\}$. This choice is repeated every X time-units, where X is exponentially distributed with rate 0.05. That is to say, with a mean of 20 seconds, the number of active CPs will change. Packet losses are not considered, i.e., every transmitted probe will eventually be answered. The value of δ_{min} has been set to 0.1, and d_{min} equals 0.5. In this scenario, the mean load of a device in

steady-state is 9.7 $probes/s$, and the variance 20.0, yielding a standard deviation of $\approx \pm 4.5$. Statistically, the probability of exceeding the nominal probe load is low. Fig. 5 depicts the device load and the number of CPs over an interval of 30 minutes (1800 s). Despite the low variance, the device load has some peaks, especially when many CPs join the network simultaneously. However, the load falls off very quickly again towards $L_{nom} = 10 = 1/\delta_{min}$ as the device rapidly incorporates the newcomers into the current schedule. It is important to realize that this is a worst-case scenario, and it makes the unrealistic assumption that all probes in a burst will eventually be replied to (no packet loss). In case of packet losses, however, which will occur in bursts due to the limited capacity of devices, the load caused by new CPs will spread better over time, since some CPs will only receive a reply after some re-probing. We can therefore expect that in practice the peaks in the device load as they appear as spikes in Fig. 5 will be a bit wider.

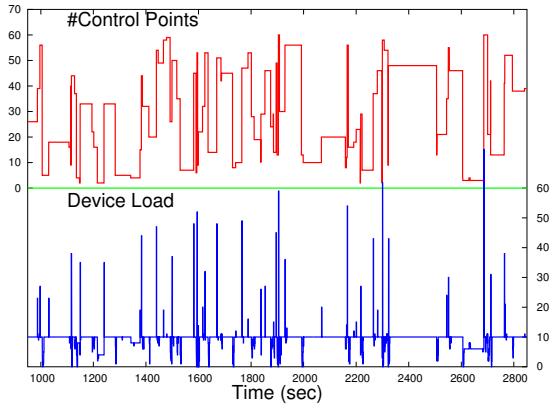

Figure 5. Load and #CPs over 30 min

6. Concluding Remarks

Our analysis has shown that the self-adaptive probe protocol SAPP suffers from a fairness problem. Some CPs can have low probing frequencies, whereas other CPs probe very fast, and even have oscillating frequencies. Faster CPs send more packets than really necessary and have a lot of computation to do in order to adjust their frequencies. This leads to a waste of computing resources and an increase of power consumption, which for e.g., handheld devices is most undesirable.

Our analysis has shown that DCPP does not suffer from the fairness problem. Moreover, the protocol is capable of keeping the load of a device within desirable limits, and—due to its intrinsic simplicity—is amenable to implementation in small computing devices.

The analysis results have been obtained using the MODEST/MÖBIUS tool suite. MODEST is a modeling language with a formal semantics [7] expressed in terms of (extended) labeled transition systems. The formality of the language allows not only for the integration with other formal analysis tools (such as model checkers), but, more importantly, is essential to carry out semantically sound simulation runs with MÖBIUS. This results in a trustworthy analysis chain (one that can be validated by means of the semantics). Standard simulation environments are risky to use instead, because they have been found to exhibit contradictory results (both quantitatively and qualitatively, i.e., difference in behavior) even in simple case studies [5].

Acknowledgment. The authors like to thank Lex Heerink and Maarten Bodlaender, both of Philips Research, for their comments on a draft of this paper.

References

[1] M. Bodlaender, J. Guidi and L. Heerink. Enhancing discovery with liveness. In: *IEEE Consumer Comm. and Netw. Conf.*, IEEE CS Press, 2004.

[2] H. Bohnenkamp, T. Courtney, D. Daly, S. Derisavi, H. Hermanns, J.-P. Katoen, V. Lam and W.H. Sanders. On integrating the Möbius and MODEST modeling tools. *DSN '03*, pp. 671–672, 2003, IEEE CS Press.

[3] H. Bohnenkamp, P.R. D'Argenio, H. Hermanns, and J.-P. Katoen. MODEST: A compositional modeling formalism for real-time and stochastic systems. CTIT Tech. Rep. 04-46, 2004.

[4] H. Bohnenkamp, H. Hermanns, J. Klaren, A. Mader, and Y. Usenko. Synthesis and stochastic assessment of schedules for lacquer production. In *QEST 2004*, pages 28–37. IEEE CS Press, Sep 2004.

[5] D. Cavin, Y. Sasson, and A. Schiper. On the accuracy of MANET simulators. In *ACM Work. Princ. Mobile Comp.*, pp. 38–43, 2002.

[6] G. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: A comprehensive study. *ACM Comput. Surv*, vol 33(4): pp. 427-469, 2001.

[7] P.R. D'Argenio, H. Hermanns, J.-P. Katoen and J. Klaren. MODEST: A modelling language for stochastic timed systems. In: *PAPM '01*, LNCS 2165: 87–104, 2001.

[8] P.R. D'Argenio, J.-P. Katoen, T.C. Ruys and G. Tretmans. The bounded retransmission protocol must be on time! In *TACAS '97*, LNCS 1217: 416–431, 1997.

[9] D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derasavi, J. Doyle, W.H. Sanders and P. Webster. The MÖBIUS framework and its implementation. *IEEE Tr. on Softw. Eng.*, **28**(10):956–970, 2002.

[10] `http://wwwhome.cs.utwente.nl/~bohnenka/liveness-model.tar.gz`

[11] M. Jelasity, W. Kowalczyk and M. van Steen. Newscast computing. Tech. Rep. IR-CS-006, Vrije Univ. Amsterdam, 2003.